Procedia Computer Science

Volume 53, 2015, Pages 167–174

2015 INNS Conference on Big Data

On the Distributed Implementation of Unsupervised Extreme Learning Machines for Big Data

Yara Rizk[1] and Mariette Awad[2]

[1] American University of Beirut, Beirut, Lebanon
yar01@aub.edu.lb

[2] American University of Beirut, Beirut, Lebanon
mariette.awad@aub.edu.lb

Abstract

The emergence of the big data problem has pushed the machine learning research community to develop unsupervised, distributed and computationally efficient learning algorithms to benefit from this data. Extreme learning machines (ELM) have gained popularity as a neuron based architecture with fast training time and good generalization. In this work, we parallelize an ELM algorithm for unsupervised learning on a distributed framework to learn clustering models from big data based on the unsupervised ELM algorithm proposed in the literature. We propose three approaches to do so: 1) Parallel US-ELM which simply distributes the data over computing nodes, 2) Hierarchical US-ELM which hierarchically clusters the data and 3) Ensemble US-ELM which is an ensemble of weak ELM models. The algorithms achieved faster training times compared to their serial counterparts and generalized better than other clustering algorithms in the literature, when tested on multiple datasets from UCI.

Keywords: Big data, Extreme Learning Machines, Unsupervised learning, Distributed computing

1 Introduction

The surge in mobile computing and social media sites in addition to the increase in memory storage capabilities has lead to the abundance of large amounts of unstructured and unlabelled data that need analysis. The most common method to alleviate the big data problem is to distribute the computations over clusters of computing nodes. Since labelling the data is expensive, developing unsupervised machine learning (ML) algorithms for big data has become a popular research area. Neural networks (NN) are popular ML algorithms composed of connected neurons, proven to be universal approximators [3, 5], that have been trained in supervised, semi-supervised and unsupervised environments. Many generations of neural networks have been developed starting from the perceptron to the more recent deep network architectures. Various learning algorithms such as backpropagation and greedy layer based training, have also

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.291

been proposed to train these networks and produced improving results. Extreme learning machines (ELM) became popular due to their fast training times compared to other NN training algorithms [7] while preserving the universal approximators property of neurons [9, 8].

Our work improves on the clustering ELM algorithm presented in [6], which is referred to as Basic US-ELM hereafter, by proposing multiple distributed implementations. Parallel US-ELM simply distributes the data over the computing nodes which map the data to the feature space, then performs the clustering algorithm. Hierarchical US-ELM hierarchically clusters the data by finding the cluster centers of the distributed data on the computing nodes, then clusters these centers on a single node. Finally, the Ensemble US-ELM implements an ensemble US-ELM which runs weak US-ELM models on the computing nodes and aggregates the votes using a majority vote scheme. Furthermore, the k-means in the serial algorithm is replaced by other algorithms and compared to the original implementation, in addition to other algorithms in the literature on multiple datasets from the UCI repository [10].

Next, we present a literature review of related works. Section 3 presents the proposed distributed ELM based algorithms for unsupervised learning. Section 4 presents a theoretical analysis of the computational complexity of US-ELM. Section 5 reports on the experimental results on multiple datasets while Section 6 concludes the paper.

2 Literature Review

Several ELM implementations on distributed frameworks have been proposed. Chen et al. parallelized an ELM ensemble classifier on MapReduce where the mapper was one ELM network and the reducer aggregated the classifiers' votes to obtain the final classification [2]. Van Heeswijk et al. presented an ensemble ELM algorithm for big data regression problems implemented on GPUs [12]. Xin et al. implemented an ELM algorithm for classification on MapReduce by distributing the Moore-Penrose generalized inverse [15]. He et al. presented a MapReduce implementation of the ELM algorithm for regression by distributing the matrix computations [4]. Bi et al. implemented a kernel ELM for classification on MapReduce by distributing the kernel and Moore-Penrose generalized inverse [1].

Zhou et al. proposed a stacked ELM targeted to improve ELM's performance on large datasets [17]. Instead of using a single layer feedforward network with a large number of neurons, the number of layers was increased and the number of neurons per layer were decreased to reduce the amount of computations on large datasets. PCA was used to reduce the number of neurons per layer and the network was iteratively trained. Wang et al. presented a parallelized ELM decision tree that used information entropy and ambiguity measures to split the nodes and apply ELM at the leafs[14]. Parallelizing the information entropy, gain, and matrix computations, allowed its application to big data classification problems.

Since online sequential ELM (OS-ELM) suffers from an inability to halt when learning from large datasets when the data does not present new information that requires model updates, Zhai et al. introduced a halting condition by dividing the data into training, validation and testing sets where the lack of improvement on the validation set accuracy was used as a termination condition and trained an ensemble of OS-ELM on large datasets [16]. Wang et al. distributed the OS-ELM computations using MapReduce by parallelizing the matrix multiplications [13].

Huang et al. proposed an unsupervised ELM algorithm, referred to as Basic US-ELM in this text, for clustering problems [6]. The algorithm transformed the data from the input space to a feature space using an ELM network and applied K-means to the embedding matrix. Basic US-ELM outperformed other clustering algorithms but did not target big data problems and was not parallelized.

Table 1: Basic US-ELM Algorithm [6]

1.	Compute the Laplacian graph matrix L from the input data
2.	Compute H using (1)
3.	Solve for the eigenvectors using (3)
4.	Compute β using (4)
5.	Compute the embedding matrix $E = H\beta$
6.	Cluster the rows of E using k-means

3 Proposed Solution

3.1 ELM Overview

Due to the universal approximation capabilities of neurons [8], neuron based architectures have been widely used. However, their popularity decreased due to their slow training times, especially as the network architecture or training data size increased. In an attempt to speed up the training time of neural networks, Huang et al. proposed ELM that could train a single hidden layer feedforward network in one step [9]. This algorithm randomly assigned input weights and analytically computed the output weights β by computing the output matrix H, shown in (1) where $g(.)$ is the activation function, x_i is a data point, w_i is an input connection weight and b_i a bias. The output weights are computed using the generalized Moore-Penrose inverse, $\beta = (H^T H)^{-1} H^T T$, where T is the target matrix in a supervised learning paradigm. Due to their fast training time, the supervised learning ELM algorithm was extended to other problems such as unsupervised learning, representational learning and others [7].

$$H = \begin{bmatrix} g(w_1 x_1 + b_1) & \dots & g(w_\ell x_1 + b_\ell) \\ \vdots & \ddots & \vdots \\ g(w_1 x_N + b_1) & \dots & g(w_\ell x_N + b_\ell) \end{bmatrix} \quad (1)$$

3.2 Unsupervised ELM (US-ELM)

Huang et al. [6] formulated the clustering problem as a manifold regularization problem with the constrained minimization problem in (2). $L = D - W \in R^{N \times N}$ is the Laplacian graph matrix computed using N training samples where $D_{ii} = \sum_{j=1}^{N} w_{ij}$ and $W = [w_{ij}], w_{ij} = e^{\frac{-||x_i - x_j||^2}{2\sigma^2}}$. $H \in R^{N \times \ell}$ is the hidden layer matrix obtained from ELM's single hidden layer feedforward NN with ℓ equal to the number of hidden layer neurons. $\beta \in R^{\ell \times m}$ is ELM's output weight matrix with m output layer neurons. γ and λ represent the eigenvalues and tuning parameters, respectively. Huang et al. proved that solving the minimization problem in (2) is equivalent to solving the generalized eigenvalue problem in (3). Since ELM is mapping the input data to an m-dimensional feature space, only m generalized eigenvectors (u or v) will be used to obtain β, as shown in (4). Once β is computed, the embedding matrix, $E = H\beta$, can be computed and used by k-means to cluster the data. K-means is randomly initialized and a grid search is performed to find the best number of clusters. The main steps in Basic US-ELM are summarized in Table 1. Although k-means performs well on small datasets, it becomes less computationally efficient as the dataset size increases [11]. Therefore, to compare the performance of US-ELM on big data problems, we propose to investigate other clustering algorithms in step 5 such as fuzzy c-means (FCM) and Gath-Geva (GG), chosen for their simple implementations.

$$\begin{aligned} min_{\beta \in R^{\ell \times m}} ||\beta||^2 + \lambda Tr(\beta^T H^T L H \beta) \\ \text{such that } \beta^T H^T H \beta = I_m \end{aligned} \tag{2}$$

$$(I_\ell + \lambda H^T L H) v = \gamma H^T H v \tag{3}$$

$$\begin{aligned} \text{if } \ell \leq N, \text{ then } \beta = [\tilde{v_2} ... \tilde{v_{m+1}}] \text{ where } \tilde{v_i} = \frac{v_i}{||Hv_i||},\ i = 2, ..., m+1 \\ \text{else } \beta = H^T [\tilde{u_2} ... \tilde{u_{m+1}}] \text{ where } \tilde{u_i} = \frac{u_i}{||HH^T u_i||},\ i = 2, ..., m+1 \end{aligned} \tag{4}$$

3.3 Distributed Unsupervised ELM

Basic US-ELM becomes more computationally expensive as the training data size increases. To reduce the computational complexity, the algorithm is distributed over multiple nodes. Based on the previous section, the US-ELM algorithm can be viewed as a two step algorithm where the first step is an input to feature space transformation and the second step is applying a clustering algorithm to the data in feature space. The first approach to parallelize this algorithm, shown in Figure 1 (left most) and referred to as Parallel US-ELM, is to distribute the feature space transformation over multiple works and apply the clustering algorithm on a single worker, assuming the data in feature space fits in memory. The same ELM network is used by all the workers to transform the subsets of data to feature space.

A second approach, shown in Figure 1 (center) and referred to as Hierarchical US-ELM, implements a hierarchical clustering algorithm based on US-ELM. The data is distributed over multiple workers that use the same ELM network, as in Parallel US-ELM, to transform the data to feature space and find the cluster centers of the data subset. Then, these centers are sent through a second round of clustering to obtain the final set of cluster heads.

Finally, we also propose a distributed ensemble US-ELM algorithm, shown in Figure 1 (right most) and referred to as Ensemble US-ELM, which implements a number of distinct ELM networks, unlike Parallel US-ELM, on the workers to transform the data and obtain the cluster heads. Then, their votes are aggregated by one worker to obtain the final cluster labels.

4 Computational Complexity Analysis

Basic US-ELM consists of five main steps. First, the Laplacian graph is computed from the training samples and requires $N^2+N^2+N^2(F+k) = O(N^2F)$ operations, where F is the number of features in input space and k is a constant. The H matrix requires $N\ell(N+k) = O(N^2\ell)$ computations, assuming a Gaussian activation function is used. Solving for the eigenvectors requires $O(N^3)$ operations. Based on (4), computing β requires $O(Nm^2\ell)$ operations when $\ell \leq N$ and $O(m(Nm\ell + N^2\ell))$ otherwise. The embedding matrix computation is the matrix multiplication of an $N \times \ell$ matrix with an $\ell \times m$ matrix and requires $O(Nm\ell)$ operations. K-means, based on Euclidean distance measure, needs $O(N_{max}NmN_c)$ during training and $O(N_{te}mN_c)$ during testing, where N_{max} is the total number of iterations needed for k-means to converge, N_{te} is the number of test samples and N_c is the number of clusters. Therefore, the total operations during training are of the order of $O(N^2F+N^3+Nm^2\ell+Nm\ell+N_{max}NmN_c)$ and $O(N^2F + N^3 + Nm^2\ell + Nm\ell + NmN_c)$ during testing.

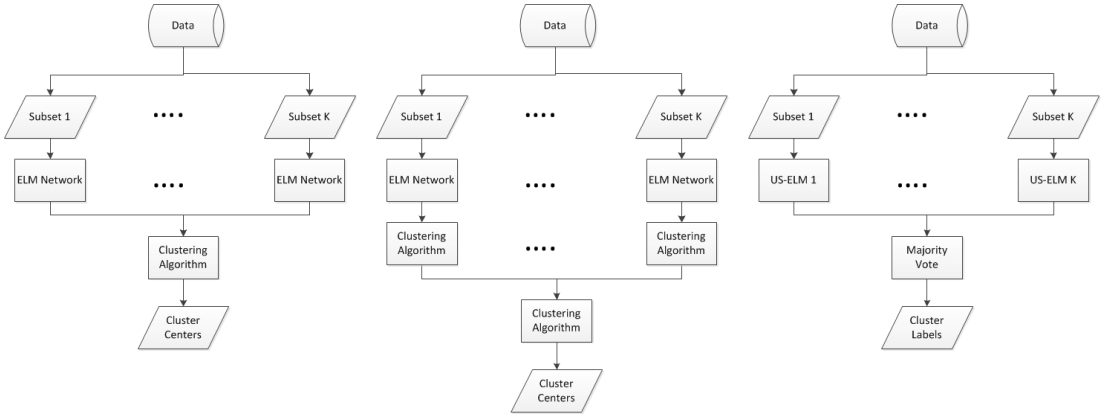

Figure 1: Block diagrams of the three distributed US-ELM algorithms: From left to right: Parallel US-ELM, Hierarchical US-ELM, Ensemble US-ELM

Table 2: Database Characteristics

Dataset	Number of Instances	Number of Attributes
Iris	150	4
Thyroid	7,198	21
Isolet	7,797	617
Letters	20,000	16
Shuttle	58,000	9
Skin Segmentation	245,057	3
Bag of Words (subset)	1,000,000	3

The memory requirements of US-ELM: $O(N^2)$ floating point numbers are needed for the Laplacian matrix, $O(N\ell)$ for the H matrix, $O(m+1)$ for the eigenvectors, $O(m\ell)$ for β, and $O(Nm)$ for E. K-means needs to keep track of the cluster centers $O(mN_c)$ and the distances between the data points and cluster centers, $O(N_cN)$. Therefore, the total memory requirements are of the order of $O(N(N+\ell+N_c)+m(\ell+N+N_c+1))$.

5 Experimental Results

5.1 Experimental Setup

The experiments were run in MATLAB R2014 on an Intel Xeon processor. The serial US-ELM variants were compared to Basic US-ELM, k-means, FCM, and GG, on multiple datasets. Parallel US-ELM, Hierarchical US-ELM and Ensemble US-ELM used k-means in the clustering step. Table 2 summarizes the characteristics of the publicly available databases from the UCI machine learning repository [10], used to validate our proposed solution based on a 4-fold cross validation scheme. A subset of the Bag of Words database was used in the experiments.

Table 3: Performance of serial US-ELM variants compared to other clustering algorithms

Database	**Iris**		**Thyroid**		**Skin Segmentation**	
Algorithm	**Number of Neurons**	**Test Accuracy**	**Number of Neurons**	**Test Accuracy**	**Number of Neurons**	**Test Accuracy**
Hierarchical US-ELM	1000	91.999	1000	**87.608**	2000	92.389
Ensemble US-ELM	1500	93.332	1000	66.715	1000	**92.970**
Basic US-ELM + k-means	500	87.322	1000	75.120	1000	90.068
Basic US-ELM + FCM	3000	**94.683**	500	56.794	1000	87.545
Basic US-ELM + GG	3000	93.332	500	72.493	1500	86.091
k-means	0	89.385	0	60.169	0	77.630
FCM	0	90.043	0	54.668	0	76.578
GG	0	91.341	0	NA	0	56.521

5.2 US-ELM Variants Performance

We compare the performance of the serial implementation of the US-ELM variants to other serial implementations of clustering algorithms: k-means, FCM and GG. In general, US-ELM based clustering algorithms performed better than the remaining clustering algorithms since they transformed the data to a feature space where the data points were better clustered. However, as the size of the datasets increased, the cost of this transformation also increased, making Basic US-ELM computationally expensive for big data problem. Comparing the US-ELM algorithms using different clustering algorithms produced comparable performances but the network architecture varied to achieve these comparable accuracies. We also notice that the GG algorithm results in a numerical error when run on the Thyroid database in input space but is able to process the data and obtain an accuracy of 72.493% when the data is transformed to the feature space using Basic US-ELM. Therefore, US-ELM's transformation allows the application of GG to datasets that would have generated singularities. Finally, comparing Hierarchical US-ELM and Ensemble US-ELM to Basic US-ELM, the former algorithms performed better on the larger datasets but worse on the smaller ones. Hierarchically clustering the data is more suitable than Parallel US-ELM when the data does not fit in the memory of one computing node and a reduction in the data is necessary. Furthermore, using different ELM networks in Ensemble US-ELM did not degrade the US-ELM's performance, results consistent with the premise of ELM algorithms [9].

5.3 Distributed Implementation Performance

The training time of US-ELM on large datasets is expensive. Examining the time spent in every step of US-ELM, as shown in Table 4, shows that the most time consuming steps are the Laplacian graph computation and ELM embedding steps. Therefore, the algorithms were distributed over multiple workers to reduce the total computational time. The number of workers was varied from 1 to 12 for all databases except for the Bag of words database which

Table 4: Serial US-ELM running time (in seconds)

Dataset	**Number of Neurons**	**Laplacian Graph**	**ELM Embedding**	**k-means**	**FCM**	**GG**
Iris	3000	0.0037	0.0685	0.0044	0.0129	0.1021
Thyroid	1000	3.2750	2.0699	0.0144	1.1082	1.8720
Isolet	1000	12.7722	75.3057	3.6640	43.2725	246.9770
Letters	1500	53.5911	11.1031	2.4630	49.5267	815.2051
Skin Segmentation	500	6548.3785	16.1122	0.8348	3.0653	9.3922

Table 5: Speedup of distributed implementations

Dataset	**Parallel US-ELM**		**Hierarchical US-ELM**		**Ensemble US-ELM**	
	Ratio of Workers	**Speedup**	**Ratio of Workers**	**Speedup**	**Ratio of Workers**	**Speedup**
Iris	2:1	1.0321	2:1	1.0654	2:1	1.5341
Thyroid	2:1	2.4483	6:1	1.5894	6:1	1.1369
Isolet	8:1	17.4416	8:1	18.3653	8:1	46.3005
Letters	8:1	23.2962	12:1	38.2719	12:1	21.9480
Shuttle	6:1	8.3644	6:1	11.3358	6:1	5.3908
Skin Segmentation	12:1	134.5725	12:1	136.6523	12:1	13.4063
Bag of Words (subset)	3:1	4.7662	3:1	6.2890	3:1	29.4171

was distributed over 8, 12 and 24 workers only because using less workers resulted in larger data partitions per worker which did not fit in a worker's memory. Table 5 reports on the highest speedup achieved for each database, where the speedup is computed using $Speedup = \frac{Time_{serial}}{Time_{distributed}}$. For small databases, such as Iris, the distribution overhead is greater than the execution time speedup causing the overall training time to increase when distributing the computations, i.e. using 2 workers had a larger speedup than using 12 workers since the size of the database does not compensate for the communication costs. However, as the size of the data increases, the speedup increases. The Skin Segmentation database had the highest speedup when using 12 workers which was approximately 135 times for the Parallel US-ELM algorithm. Based on the obtained results, we notice that to maximize the potential of the proposed algorithms, the datasets should be large. We also notice that as the database increases, the speedup decreased. This is evident when comparing the Skin Segmentation database which is almost a quarter of the size of the Bag of Words database but achieved higher speedup, approximately 135 vs. 4.78 respectively for the Parallel US-ELM algorithm. This is due to the suboptimal memory management because the implementations load all the data into a worker's memory. As the size of the database increases, the subset assigned to each worker also increases when the number of workers is kept constant. Therefore, the memory accesses reduce the achievable speedup. To overcome this issue, the implementations should efficiently utilize available memory or the number of workers should be increased.

6 Conclusion

US-ELM algorithms performed better than other clustering algorithms on big data when tested on multiple datasets. The three distributed implementations of US-ELM lead to faster training times than the serial implementation, making them suitable for big data problems. The clustering performances of the proposed algorithms were comparable and the choice of the most suitable algorithm was database dependent. Future work will parallelize the matrix operations of the algorithm as well as investigate other clustering algorithms than k-means, FCM and GG.

References

[1] Xin Bi, Xiangguo Zhao, Guoren Wang, Pan Zhang, and Chao Wang. Distributed extreme learning machine with kernels based on MapReduce. *Neurocomputing*, 149:456–463, 2015.

[2] Jiaoyan Chen, Guozhou Zheng, and Huajun Chen. ELM-MapReduce: MapReduce accelerated extreme learning machine for big spatial data analysis. In *10th International Conference on Control and Automation (ICCA)*, pages 400–405. IEEE, 2013.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[4] Qing He, Tianfeng Shang, Fuzhen Zhuang, and Zhongzhi Shi. Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 102:52–58, 2013.

[5] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[6] G Huang, S Song, JN Gupta, and C Wu. Semi-supervised and unsupervised extreme learning machines. *IEEE transactions on cybernetics*, 44(12):2405–2417, 2014.

[7] Gao Huang, Guang Bin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, 2015.

[8] Guang Bin Huang, Lei Chen, and Chee Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *Transactions on Neural Networks*, 17(4):879–892, 2006.

[9] Guang Bin Huang, Qin Yu Zhu, and Chee Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 985–990. IEEE, 2004.

[10] M. Lichman. UCI machine learning repository, 2013.

[11] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2000.

[12] Mark Van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74(16):2430–2437, 2011.

[13] Botao Wang, Shan Huang, Junhao Qiu, Yu Liu, and Guoren Wang. Parallel online sequential extreme learning machine based on MapReduce. *Neurocomputing*, 149:224–232, 2015.

[14] Ran Wang, Yu Lin He, Chi Yin Chow, Fang Fang Ou, and Jian Zhang. Learning ELM-tree from big data based on uncertainty reduction. *Fuzzy Sets and Systems*, 258:79–100, 2015.

[15] Junchang Xin, Zhiqiong Wang, Chen Chen, Linlin Ding, Guoren Wang, and Yuhai Zhao. ELM*: distributed extreme learning machine with MapReduce. *World Wide Web*, 17(5):1189–1204, 2014.

[16] Junhai Zhai, Jinggeng Wang, and Xizhao Wang. Ensemble online sequential extreme learning machine for large data set classification. In *International Conference on Systems, Man and Cybernetics*, pages 2250–2255. IEEE, 2014.

[17] H Zhou, GB Huang, Z Lin, H Wang, and YC Soh. Stacked extreme learning machines. *IEEE transactions on cybernetics*, PP(99):1–13, 2014.